

Operation System – COSE341

실습 1 과제 레포트 – System Call, Process, and Thread

고려대학교 컴퓨터학과 2017320108

고재영

개발 환경 : Oracle VM VirtualBox, Linux, Ubuntu 18.04.5

과제 만기일 : 2022/04/18 10:29 AM

유의 사항 : 간단한 설명과 그림판을 이용해 주요부분을 빨간색으로 강조한 스크린샷

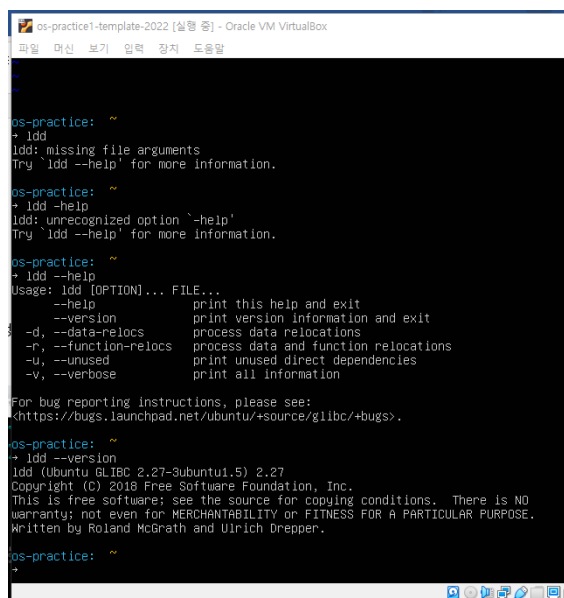
제출 날짜 : 2022.04.12

수정 후 최종 제출 날짜 : 2022.04.14

[1] - 과제 1 -1. 시스템 콜 과정 이해하기

● 사용자의 C언어 프로그램에서 read 함수가 발생되었다는 가정을 했을 때, read 시스템 콜이 호출되는 과정을 설명한다.

User mode에서 read함수가 포함된 c언어로 작성된 프로그램을 실행한다. 이 때, read함수는 프로그램 내에 파일의 내용을 읽는 함수이다. Read 함수는 GNU C 라이브러리를 사용하는데, 현재 내 환경에서의 이용하고 있는 c 라이브러리는 GLIBC 2.27-3ubuntu1.5임을 알 수 있다.



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  편집  보기  입력  장치  도움말

os-practice: ~
→ ldd
ldd: missing file arguments
Try 'ldd --help' for more information.

os-practice: ~
→ ldd -help
ldd: unrecognized option '-help'
Try 'ldd --help' for more information.

os-practice: ~
→ ldd --help
Usage: ldd [OPTION]... FILE...
  --help                print this help and exit
  --version             print version information and exit
  -d, --data-relocs     process data relocations
  -r, --function-relocs process data and function relocations
  -u, --unused          print unused direct dependencies
  -v, --verbose         print all information

For bug reporting instructions, please see:
<https://bugs.launchpad.net/ubuntu/+source/glibc/+bugs>.

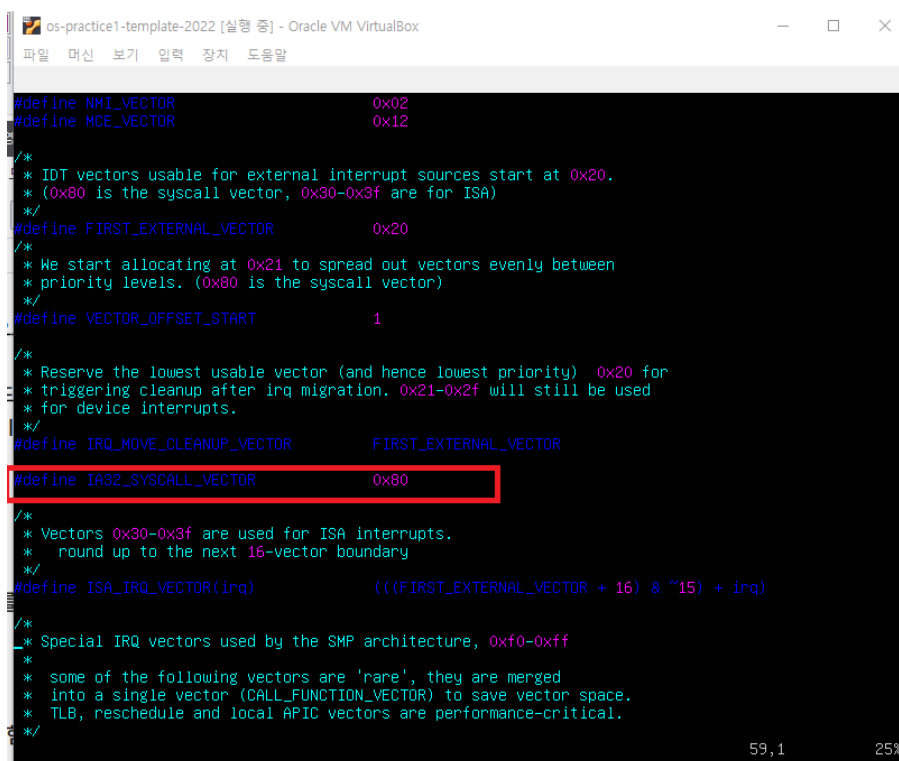
os-practice: ~
→ ldd --version
ldd (Ubuntu GLIBC 2.27-3ubuntu1.5) 2.27
Copyright (c) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Written by Roland McGrath and Ulrich Drepper.

os-practice: ~
→
```

C 라이브러리에서 read 함수의 어셈블리 코드를 찾아보면,

```
Read() {  
  
    movl $3, %eax  
  
    movl $0, %ebx  
  
    movl $resb, %ecx  
  
    movl $200, %edx  
  
    int $0x80  
  
}
```

위와 같은 코드를 가지는 것을 알 수 있다. Movl 어셈블리는 특정 레지스터에 특정 값을 넣는 연산을 하고, 이 줄의 5번째인 "int \$0x80" 연산은 0x80번, 즉 decimal로 128번째의 offset을 갖는 interrupt를 발생시킨다. 이 때, CPU는 User mode에서 Kernel mode로 전환되어 사용자 영역 뿐만 아니라 커널 영역 memory에 접근 가능하게 되며 이 interrupt를 처리한다. 커널 메모리에는 IDT(Interrupt Descriptor Table)라는 테이블 형태의 자료구조를 가지는데, 이 테이블에서 offset이 0x80인, 즉 128번째의 interrupt를 발생시키므로, 이에 해당하는 system_call()이란 로직을 발생시킨다. IDT의 system_call()은 실제 시스템 콜을 처리하는 함수를 저장한 테이블인 system call table을 가리키는 포인터이다.



```
#define NM1_VECTOR          0x02  
#define MCE_VECTOR          0x12  
  
/*  
 * IDT vectors usable for external interrupt sources start at 0x20.  
 * (0x80 is the syscall vector, 0x30-0x3f are for ISA)  
 */  
#define FIRST_EXTERNAL_VECTOR 0x20  
/*  
 * We start allocating at 0x21 to spread out vectors evenly between  
 * priority levels. (0x80 is the syscall vector)  
 */  
#define VECTOR_OFFSET_START 1  
  
/*  
 * Reserve the lowest usable vector (and hence lowest priority) 0x20 for  
 * triggering cleanup after irq migration. 0x21-0x2f will still be used  
 * for device interrupts.  
 */  
#define IRQ_MOVE_CLEANUP_VECTOR FIRST_EXTERNAL_VECTOR  
#define IA32_SYSCALL_VECTOR 0x80  
  
/*  
 * Vectors 0x30-0x3f are used for ISA interrupts.  
 * round up to the next 16-vector boundary  
 */  
#define ISA_IRQ_VECTOR(irq) (((FIRST_EXTERNAL_VECTOR + 16) & ~15) + irq)  
  
/*  
 * Special IRQ vectors used by the SMP architecture, 0xf0-0xff  
 * some of the following vectors are 'rare', they are merged  
 * into a single vector (CALL_FUNCTION_VECTOR) to save vector space.  
 * TLB, reschedule and local APIC vectors are performance-critical.  
 */
```

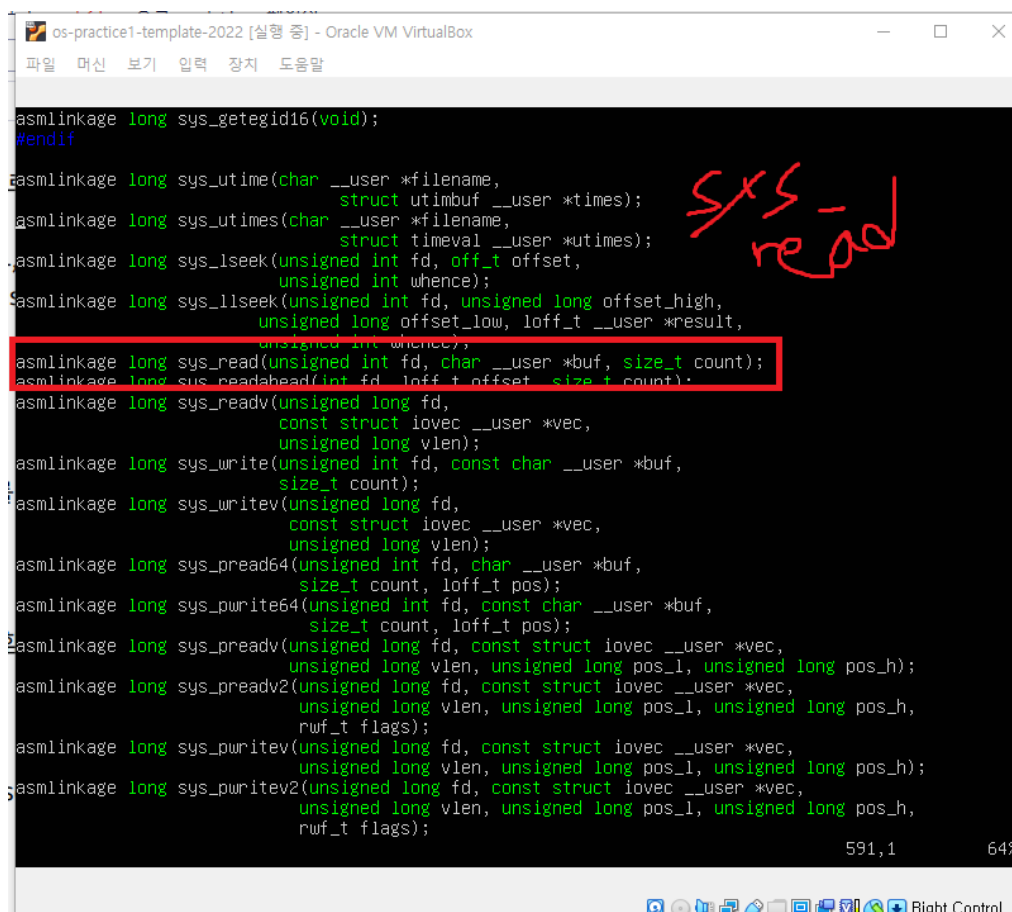
아래의 두 스크린샷은, 직접 리눅스 커널에서

vi /usr/src/linux/include/linux/syscalls.h

vi /usr/src/linux/arch/x86/entry/syscalls/syscall_64.tbl

로 접근하여 찾은 것이다.

다음은 보면 알 수 있다시피, system call table의 0번째가 read 시스템 콜이고, 실제로 이를 처리하는 함수는 sys_read()임을 알 수 있다. 위에서 살펴본 어셈블리 코드와 아래 스크린샷으로 미루어 보면, 4번의 movl 연산은 sys_read의 3개의 매개변수를 전달해주고, "movl \$0, %ebx"가 바로 system call table의 0번째 인덱스를 부르는 것을 알 수 있다.



```
asmlinkage long sys_getegid16(void);
#endif

asmlinkage long sys_utime(char __user *filename,
                          struct utimbuf __user *times);
asmlinkage long sys_utimes(char __user *filename,
                           struct timeval __user *utimes);
asmlinkage long sys_lseek(unsigned int fd, off_t offset,
                          unsigned int whence);
asmlinkage long sys_llseek(unsigned int fd, unsigned long offset_high,
                           unsigned long offset_low, loff_t __user *result,
                           unsigned int whence);
asmlinkage long sys_read(unsigned int fd, char __user *buf, size_t count);
asmlinkage long sys_readahead(int fd, loff_t offset, size_t count);
asmlinkage long sys_readv(unsigned long fd,
                          const struct iovec __user *vec,
                          unsigned long vlen);
asmlinkage long sys_write(unsigned int fd, const char __user *buf,
                          size_t count);
asmlinkage long sys_writew(unsigned long fd,
                           const struct iovec __user *vec,
                           unsigned long vlen);
asmlinkage long sys_pread64(unsigned int fd, char __user *buf,
                           size_t count, loff_t pos);
asmlinkage long sys_pwrite64(unsigned int fd, const char __user *buf,
                             size_t count, loff_t pos);
asmlinkage long sys_preadv(unsigned long fd, const struct iovec __user *vec,
                           unsigned long vlen, unsigned long pos_l, unsigned long pos_h);
asmlinkage long sys_preadv2(unsigned long fd, const struct iovec __user *vec,
                             unsigned long vlen, unsigned long pos_l, unsigned long pos_h,
                             rwf_t flags);
asmlinkage long sys_pwritev(unsigned long fd, const struct iovec __user *vec,
                             unsigned long vlen, unsigned long pos_l, unsigned long pos_h);
asmlinkage long sys_pwritev2(unsigned long fd, const struct iovec __user *vec,
                              unsigned long vlen, unsigned long pos_l, unsigned long pos_h,
                              rwf_t flags);
```

os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox

파일 머신 보기 입력 장치 도움말

```
# 64-bit system call numbers and entry vectors
#
# The format is:
# <number> <abi> <name> <entry point>
#
# The abi is "common", "64" or "x32" for this file.
0      common  read      sys_read
1      common  write     sys_write
2      common  open      sys_open
3      common  close     sys_close
4      common  stat      sys_newstat
5      common  fstat     sys_newfstat
6      common  lstat     sys_newlstat
7      common  poll      sys_poll
8      common  lseek     sys_lseek
9      common  mmap      sys_mmap
10     common  mprotect sys_mprotect
11     common  munmap    sys_munmap
12     common  brk       sys_brk
13     64      rt_sigaction sys_rt_sigaction
14     common  rt_sigprocmask sys_rt_sigprocmask
15     64      rt_sigreturn sys_rt_sigreturn/ptregs
16     64      ioctl     sys_ioctl
17     common  pread64    sys_pread64
18     common  pwrite64  sys_pwrite64
19     64      readv     sys_readv
20     64      writev    sys_writev
21     common  access    sys_access
22     common  pipe      sys_pipe
23     common  select    sys_select
24     common  sched_yield sys_sched_yield
25     common  mremap    sys_mremap
26     common  msync     sys_msync
27     common  mincore    sys_mincore
```

0번

1,1

[1] - 과제 1-2. 새로운 시스템 콜 추가하기

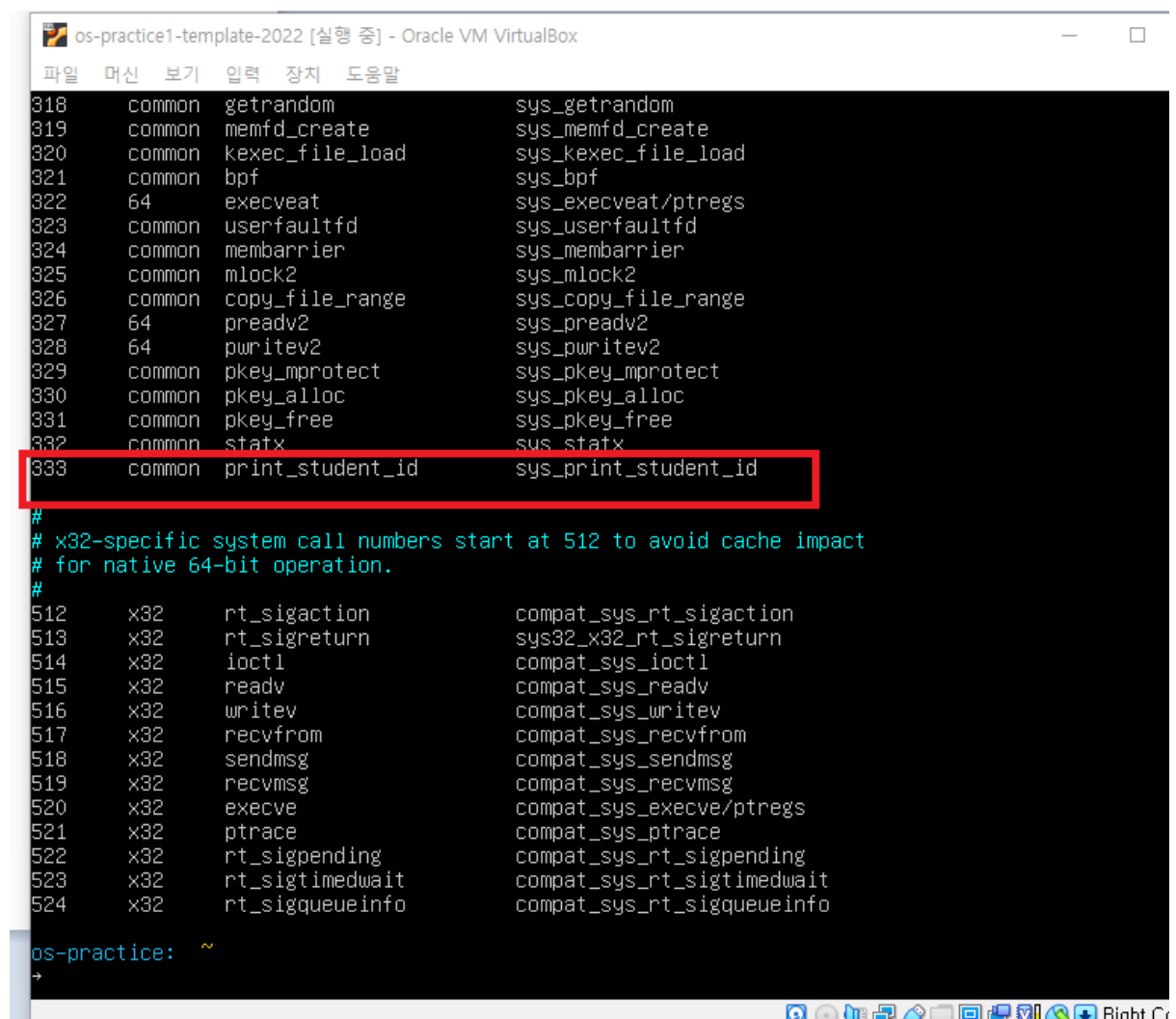
자신의 학번을 `printk` 함수를 사용해 출력하는 시스템 콜

시스템 콜 이름: `print_student_id`

호출되는 함수 이름: `sys_print_student_id(void)`

1) 시스템 콜 할당

System call은 0번 `sys_read`부터 332번 `sys_statx`까지 주어져 있다. 따라서 새로운 시스템 콜 `print_student_id`를 `sys_print_student_id`라는 이름으로 호출되도록 시스템 콜을 할당한다.



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
318  common  getrandom      sys_getrandom
319  common  memfd_create   sys_memfd_create
320  common  kexec_file_load sys_kexec_file_load
321  common  bpf            sys_bpf
322  64      execveat       sys_execveat/ptregs
323  common  userfaultfd    sys_userfaultfd
324  common  membarrier     sys_membarrier
325  common  mlock2         sys_mlock2
326  common  copy_file_range sys_copy_file_range
327  64      preadv2        sys_preadv2
328  64      pwritev2       sys_pwritev2
329  common  pkey_mprotect  sys_pkey_mprotect
330  common  pkey_alloc     sys_pkey_alloc
331  common  pkey_free      sys_pkey_free
332  common  statx          sys_statx
333  common  print_student_id sys_print_student_id
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512  x32     rt_sigaction   compat_sys_rt_sigaction
513  x32     rt_sigreturn   sys32_x32_rt_sigreturn
514  x32     ioctl          compat_sys_ioctl
515  x32     readv          compat_sys_readv
516  x32     writev         compat_sys_writev
517  x32     recvfrom       compat_sys_recvfrom
518  x32     sendmsg        compat_sys_sendmsg
519  x32     recvmsg        compat_sys_recvmsg
520  x32     execve         compat_sys_execve/ptregs
521  x32     ptrace         compat_sys_ptrace
522  x32     rt_sigpending  compat_sys_rt_sigpending
523  x32     rt_sigtimedwait compat_sys_rt_sigtimedwait
524  x32     rt_sigqueueinfo compat_sys_rt_sigqueueinfo
os-practice: ~
→
```

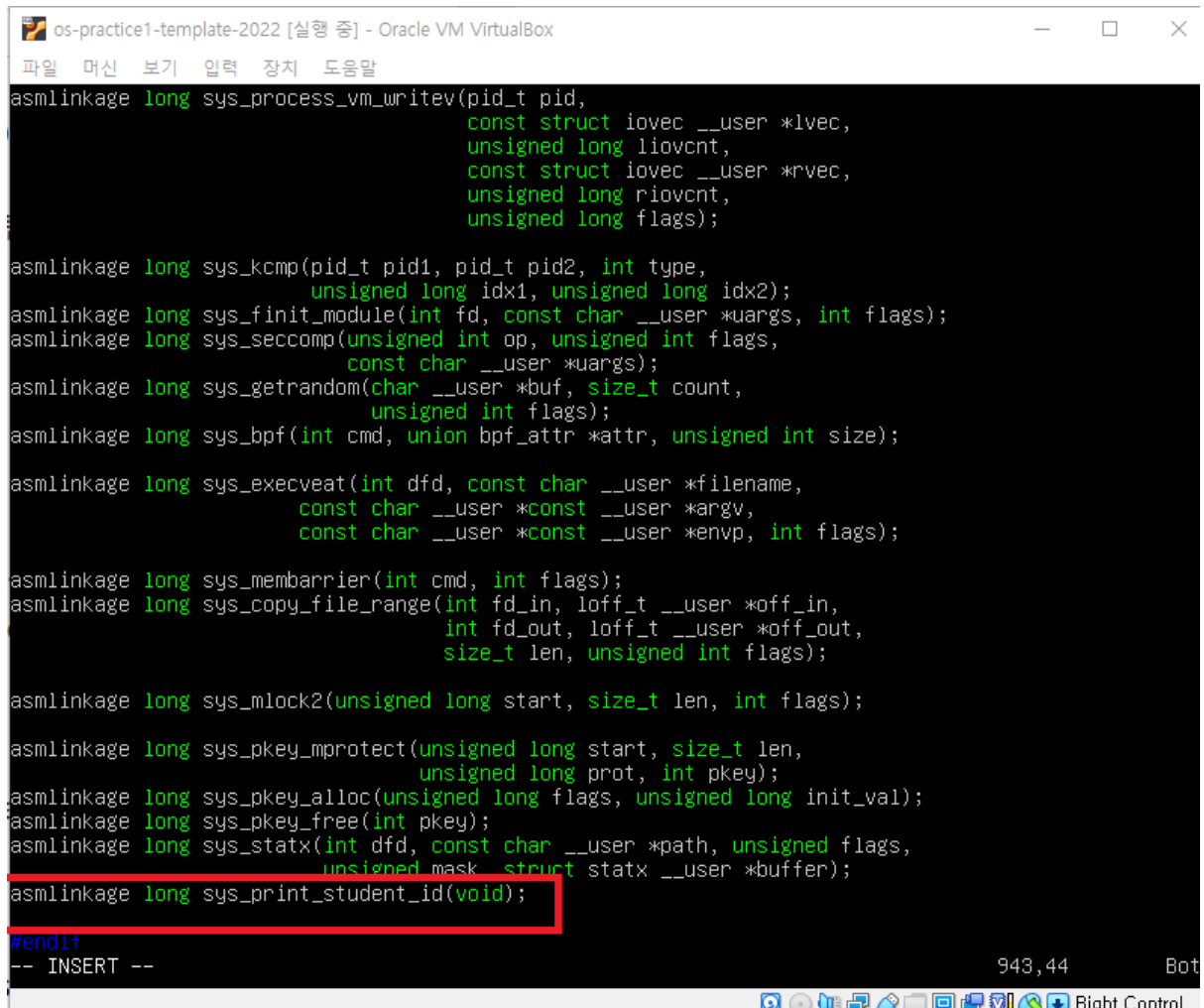
2) 시스템 콜 함수 구현

주어진 조건은 `printf` 함수를 이용하여, 나의 학번을 출력하는 함수를 만드는 것이다. "My student id is 20XXXXXXXX"를 출력하도록 명시하였으므로, 이에 따라 나의 학번인 2017320108을 출력하는 시스템 콜 함수를 `new_syscall.c`에 구현한다.

[illegible]

3) 시스템 콜 함수 선언

여타 다른 시스템 콜과 마찬가지로 long형으로 선언하며, 전달받는 매개변수는 없기 때문에 void 매개변수로 선언한다.



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
asmlinkage long sys_process_vm_writev(pid_t pid,
                                     const struct iovec __user *lvec,
                                     unsigned long liovcnt,
                                     const struct iovec __user *rvec,
                                     unsigned long riovcnt,
                                     unsigned long flags);
asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                        unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                             unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);
asmlinkage long sys_execveat(int dfd, const char __user *filename,
                             const char __user *const __user *argv,
                             const char __user *const __user *envp, int flags);
asmlinkage long sys_membarrier(int cmd, int flags);
asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,
                                    int fd_out, loff_t __user *off_out,
                                    size_t len, unsigned int flags);
asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);
asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len,
                                   unsigned long prot, int pkey);
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,
                          unsigned mask, struct statx __user *buffer);
asmlinkage long sys_print_student_id(void);
#endit
-- INSERT --
943,44 Bot
```

추가적으로, 이제 실행에 옮기기 전에, 오브젝트 파일을 makefile에 추가해줘야 하므로, /usr/src/linux/kernel/Makefile에 다음과 같이 new_syscall.o를 추가한다.

```
# SPDX-License-Identifier: GPL-2.0
#
# Makefile for the linux kernel.
#

obj-y      = fork.o exec_domain.o panic.o \
             cpu.o exit.o softirq.o resource.o \
             sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
             signal.o sys.o umh.o workqueue.o pid.o task_work.o \
             extable.o params.o \
             kthread.o sys_ni.o nsproxy.o \
             notifier.o ksysfs.o cred.o reboot.o \
             async.o range.o smpboot.o ucount.o new_syscall.o

obj-$(CONFIG_MODULES) += kmod.o
obj-$(CONFIG_MULTIUSER) += groups.o

ifdef CONFIG_FUNCTION_TRACER
# Do not trace internal ftrace files
CFLAGS_REMOVE_irq_work.o = $(CC_FLAGS_FTRACE)
endif

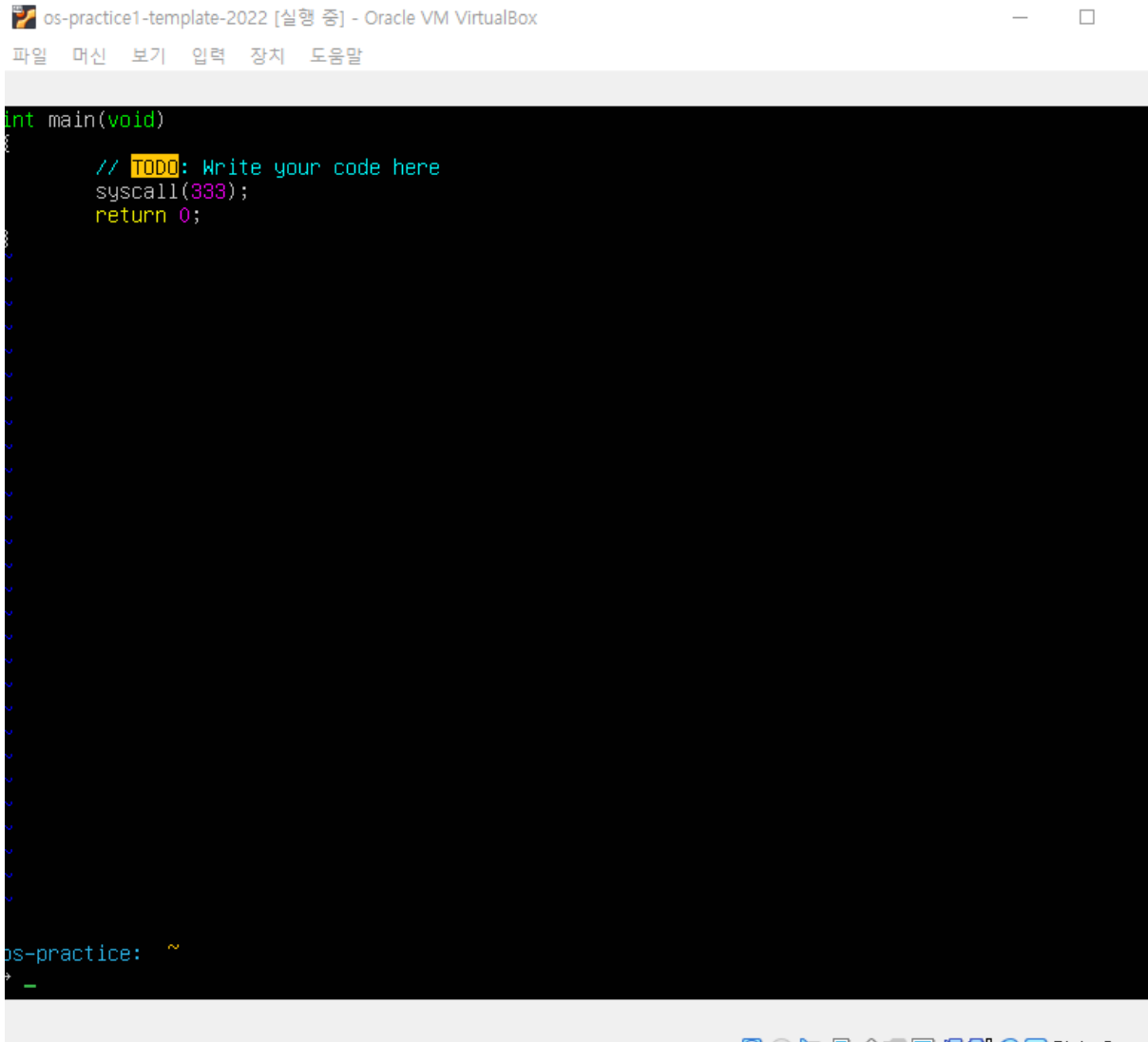
# Prevents flicker of uninteresting __do_softirq()/__local_bh_disable_ip()
# in coverage traces.
KCOV_INSTRUMENT_softirq.o := n
# These are called from save_stack_trace() on slub debug path,
# and produce insane amounts of uninteresting coverage.
KCOV_INSTRUMENT_module.o := n
KCOV_INSTRUMENT_extable.o := n
# Don't self-instrument.
KCOV_INSTRUMENT_kcov.o := n
KASAN_SANITIZE_kcov.o := n

# cond_syscall is currently not LTO compatible
CFLAGS_sys_ni.o = $(DISABLE_LTO)

"Makefile" 125L, 4083C                                     3,32
```


4) 사용자 영역 프로그램 작성

이제, 새롭게 추가한 시스템 콜을 추가하였으므로, 사용자 영역 프로그램인 assignment.c에서 해당 시스템 콜을 호출하도록 한다. 333번에 추가하였으므로, system call table의 333번째 시스템 콜을 호출하도록 `syscall(333);`이라고 작성한다.



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

int main(void)
{
    // TODO: Write your code here
    syscall(333);
    return 0;
}

os-practice: ~
```

5) 실행 및 결과

일반 사용자가 root의 권한을 부여받아 실행하는 sudo 명령어로 모듈을 설치하여 실행한다.

```
Cd /usr/src/linux
```

```
Sudo make -j 4
```

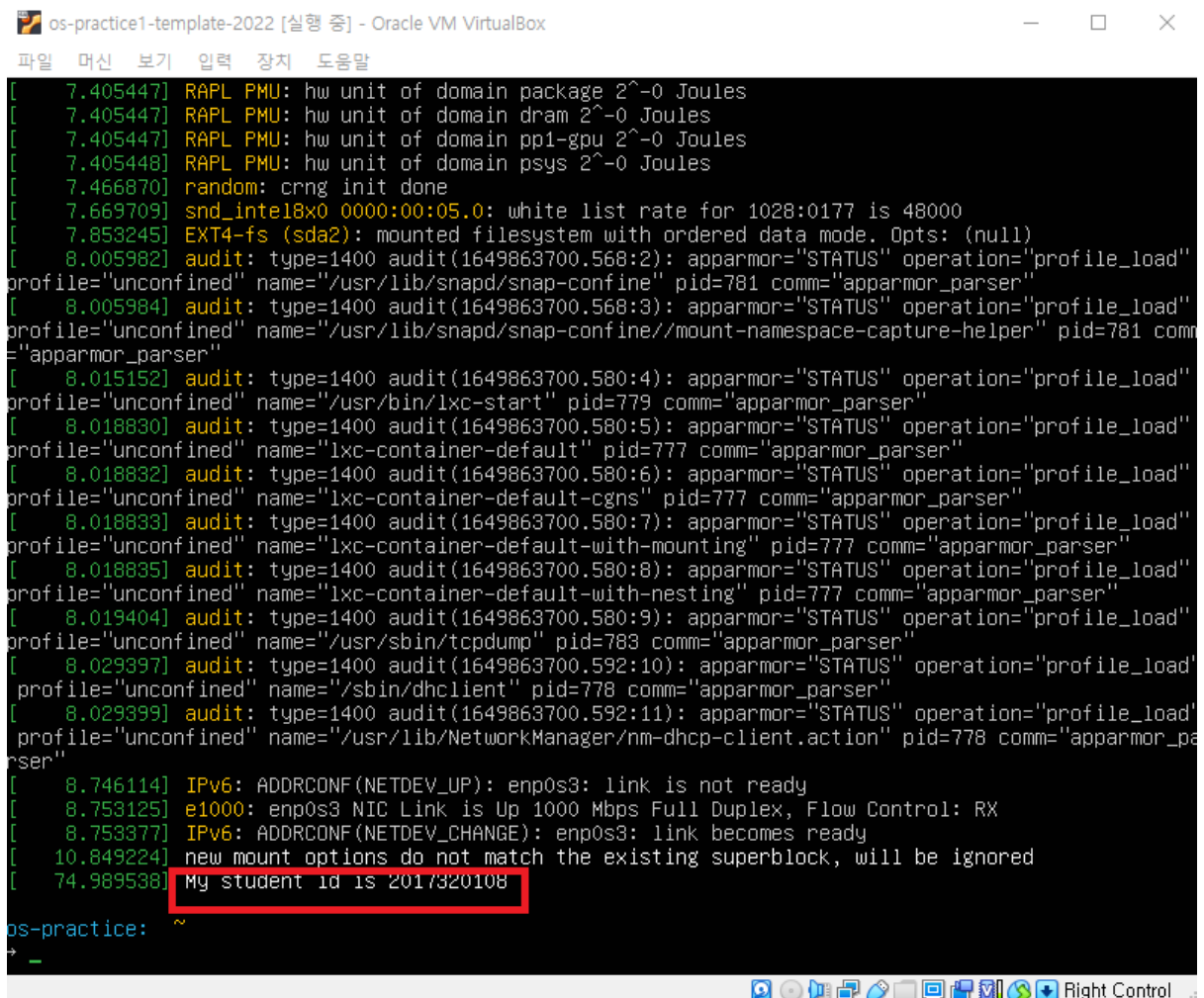
```
Sudo make modules_install
```

```
Sudo make install
```

```
Sudo reboot
```

이 때, sudo로 root 사용자가 아닌, guest로 접속했으므로, 요구하는 비밀번호를 입력해야 진행가능하다.

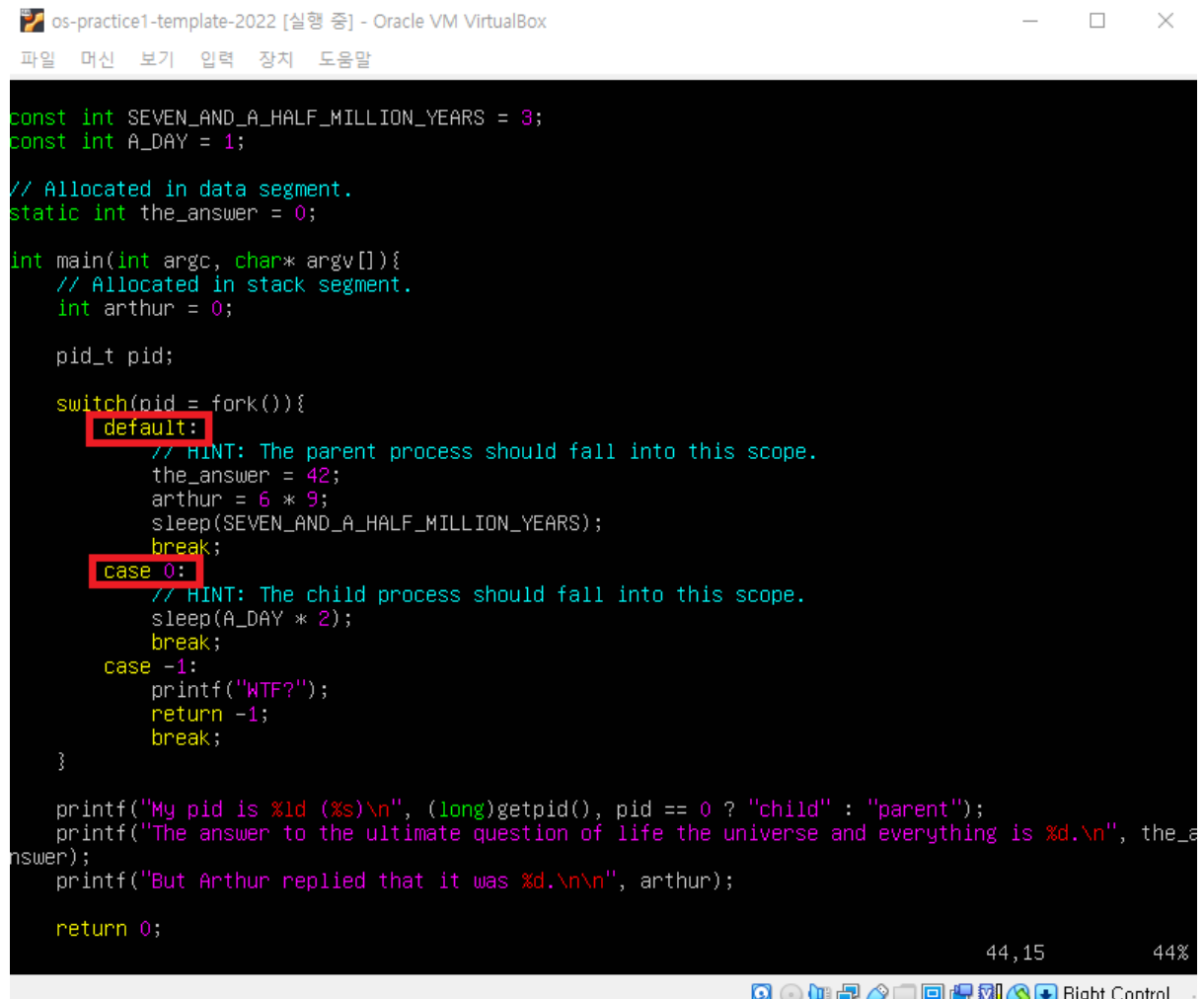
마지막으로, 이 일련의 과정들을 모두 거치고 나면, 다음과 같이 제대로 결과가 출력됨을 알 수 있다.



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
[ 7.405447] RAPL PMU: hw unit of domain package 2^-0 Joules
[ 7.405447] RAPL PMU: hw unit of domain dram 2^-0 Joules
[ 7.405447] RAPL PMU: hw unit of domain pp1-gpu 2^-0 Joules
[ 7.405448] RAPL PMU: hw unit of domain psys 2^-0 Joules
[ 7.466870] random: crng init done
[ 7.669709] snd_intel8x0 0000:00:05.0: white list rate for 1028:0177 is 48000
[ 7.853245] EXT4-fs (sda2): mounted filesystem with ordered data mode. Opts: (null)
[ 8.005982] audit: type=1400 audit(1649863700.568:2): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="/usr/lib/snapd/snap-confine" pid=781 comm="apparmor_parser"
[ 8.005984] audit: type=1400 audit(1649863700.568:3): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="/usr/lib/snapd/snap-confine/mount-namespace-capture-helper" pid=781 comm="apparmor_parser"
[ 8.015152] audit: type=1400 audit(1649863700.580:4): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="/usr/bin/lxc-start" pid=779 comm="apparmor_parser"
[ 8.018830] audit: type=1400 audit(1649863700.580:5): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="lxc-container-default" pid=777 comm="apparmor_parser"
[ 8.018832] audit: type=1400 audit(1649863700.580:6): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="lxc-container-default-cgns" pid=777 comm="apparmor_parser"
[ 8.018833] audit: type=1400 audit(1649863700.580:7): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="lxc-container-default-with-mounting" pid=777 comm="apparmor_parser"
[ 8.018835] audit: type=1400 audit(1649863700.580:8): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="lxc-container-default-with-nesting" pid=777 comm="apparmor_parser"
[ 8.019404] audit: type=1400 audit(1649863700.580:9): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="/usr/sbin/tcpdump" pid=783 comm="apparmor_parser"
[ 8.029397] audit: type=1400 audit(1649863700.592:10): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="/sbin/dhclient" pid=778 comm="apparmor_parser"
[ 8.029399] audit: type=1400 audit(1649863700.592:11): apparmor="STATUS" operation="profile_load"
profile="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-client.action" pid=778 comm="apparmor_parser"
[ 8.746114] IPv6: ADDRCONF(NETDEV_UP): enp0s3: link is not ready
[ 8.753125] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 8.753377] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[ 10.849224] new mount options do not match the existing superblock, will be ignored
[ 74.989538] My student id is 2017320108
os-practice: ~
+ -
```

[2] - 과제 2. 프로세스, 스레드 실습

프로세스 #01



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

const int SEVEN_AND_A_HALF_MILLION_YEARS = 3;
const int A_DAY = 1;

// Allocated in data segment.
static int the_answer = 0;

int main(int argc, char* argv[]){
    // Allocated in stack segment.
    int arthur = 0;

    pid_t pid;

    switch(pid = fork()){
        default:
            // HINT: The parent process should fall into this scope.
            the_answer = 42;
            arthur = 6 * 9;
            sleep(SEVEN_AND_A_HALF_MILLION_YEARS);
            break;
        case 0:
            // HINT: The child process should fall into this scope.
            sleep(A_DAY * 2);
            break;
        case -1:
            printf("WTF?");
            return -1;
            break;
    }

    printf("My pid is %ld (%s)\n", (long) getpid(), pid == 0 ? "child" : "parent");
    printf("The answer to the ultimate question of life the universe and everything is %d.\n", the_answer);
    printf("But Arthur replied that it was %d.\n\n", arthur);

    return 0;
}
```

Child process의 pid = 0,

Parent process의 pid > 0이므로 위와 같이 코드를 작성한다.

```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

os-practice: ~
→ ls
assignment assignment.c Process-Exercises Threads-Exercises

os-practice: ~
→ cd Process-Exercises
/home/guest/Process-Exercises

os-practice: ~/Process-Exercises
→ ls
exercise LICENSE quiz

os-practice: ~/Process-Exercises
→ cd quiz
/home/guest/Process-Exercises/quiz

os-practice: ~/Process-Exercises/quiz
→ cd 01
/home/guest/Process-Exercises/quiz/01

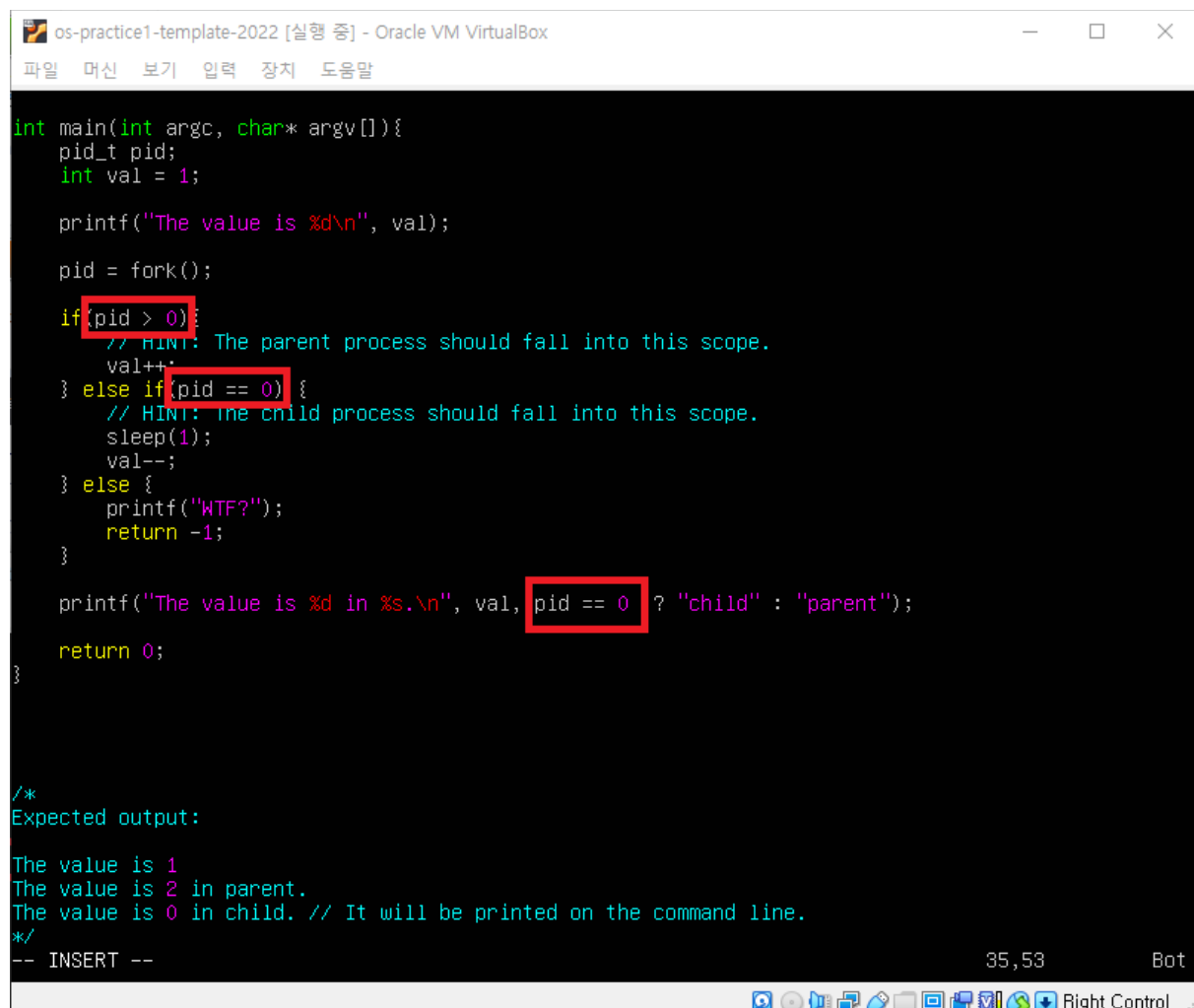
os-practice: ~/.../quiz/01
→ ls
main main.c

os-practice: ~/.../quiz/01
→ ./main
My pid is 1677 (child)
The answer to the ultimate question of life the universe and everything is 0.
But Arthur replied that it was 0.

My pid is 1676 (parent)
The answer to the ultimate question of life the universe and everything is 42.
But Arthur replied that it was 54.

os-practice: ~/.../quiz/01
→ _
```

프로세스 #02



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

int main(int argc, char* argv[]){
    pid_t pid;
    int val = 1;

    printf("The value is %d\n", val);

    pid = fork();

    if(pid > 0){
        // HINT: The parent process should fall into this scope.
        val++;
    } else if(pid == 0){
        // HINT: the child process should fall into this scope.
        sleep(1);
        val--;
    } else {
        printf("WTF?");
        return -1;
    }

    printf("The value is %d in %s.\n", val, pid == 0 ? "child" : "parent");

    return 0;
}

/*
Expected output:
The value is 1
The value is 2 in parent.
The value is 0 in child. // It will be printed on the command line.
*/
-- INSERT --
```

01번 문제와 마찬가지로의 이유이며, 삼항연산자에 들어갈 조건문은 `pid == 0`

os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox

파일 머신 보기 입력 장치 도움말

```
printf("The value is %d\n", val);

pid = fork();

if(pid > 0){
    // HINT: The parent process should fall into this scope.
    val++;
} else if(pid == 0) {
    // HINT: The child process should fall into this scope.
    sleep(1);
    val--;
} else {
    printf("WTF?");
    return -1;
}

printf("The value is %d in %s.\n", val, pid == 0 ? "child" : "parent");

return 0;
}
```

os-practice: ~/.../quiz/02
→ gcc -o main main.c

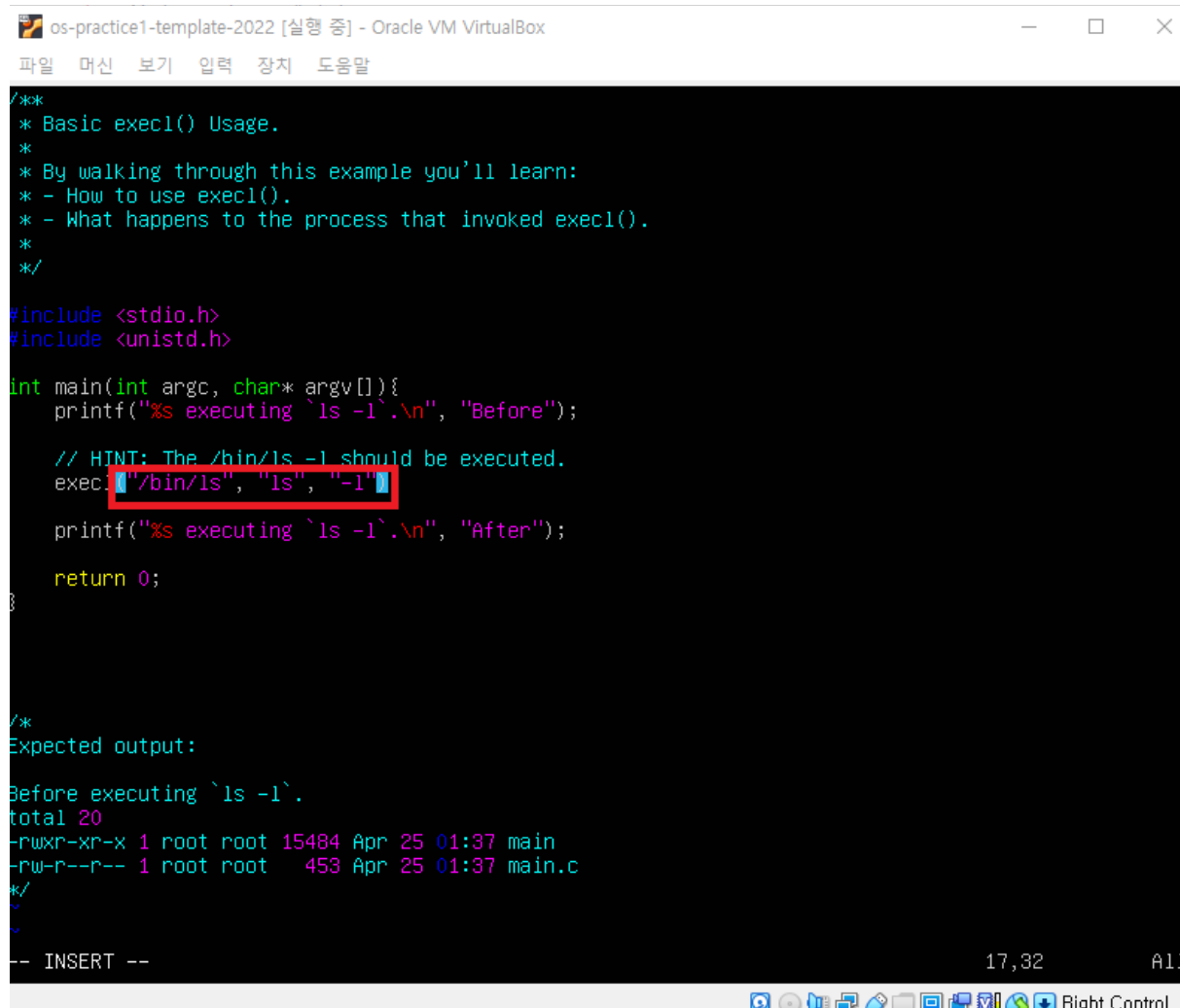
os-practice: ~/.../quiz/02
→ ./main
The value is 1
The value is 2 in parent.

os-practice: ~/.../quiz/02
→ The value is 0 in child.

os-practice: ~/.../quiz/02
→

Right Control

프로세스 #03



The screenshot shows a C program in a terminal window titled "os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox". The program demonstrates the use of `execl()` to execute `ls -l`. The code includes `<stdio.h>` and `<unistd.h>`. The `main` function prints "Before", calls `execl("/bin/ls", "ls", "-l", NULL)` (the arguments are highlighted with a red box), and then prints "After". The output shows the directory listing for the current directory, including the `main` process and the `main.c` file. The terminal also shows the expected output and some system information.

```
/*  
 * Basic execl() Usage.  
 */  
 * By walking through this example you'll learn:  
 * - How to use execl().  
 * - What happens to the process that invoked execl().  
 */  
*/  
  
#include <stdio.h>  
#include <unistd.h>  
  
int main(int argc, char* argv[]){  
    printf("%s executing `ls -l`.\n", "Before");  
  
    // HINT: The /bin/ls -l should be executed.  
    execl("/bin/ls", "ls", "-l", NULL);  
  
    printf("%s executing `ls -l`.\n", "After");  
  
    return 0;  
}  
  
/*  
Expected output:  
Before executing `ls -l`.  
total 20  
-rwxr-xr-x 1 root root 15484 Apr 25 01:37 main  
-rw-r--r-- 1 root root 453 Apr 25 01:37 main.c  
*/  
-- INSERT --  
17,32 All
```

Execl 연산자에 알맞은 매개변수, 마지막 항은 어차피 NULL 값이므로 삼항까지만 작성했다.

```
    return 0;
}

/*
Expected output:

Before executing `ls -l`.
total 20
-rwxr-xr-x 1 root root 15484 Apr 25 01:37 main
-rw-r--r-- 1 root root  453 Apr 25 01:37 main.c
*/

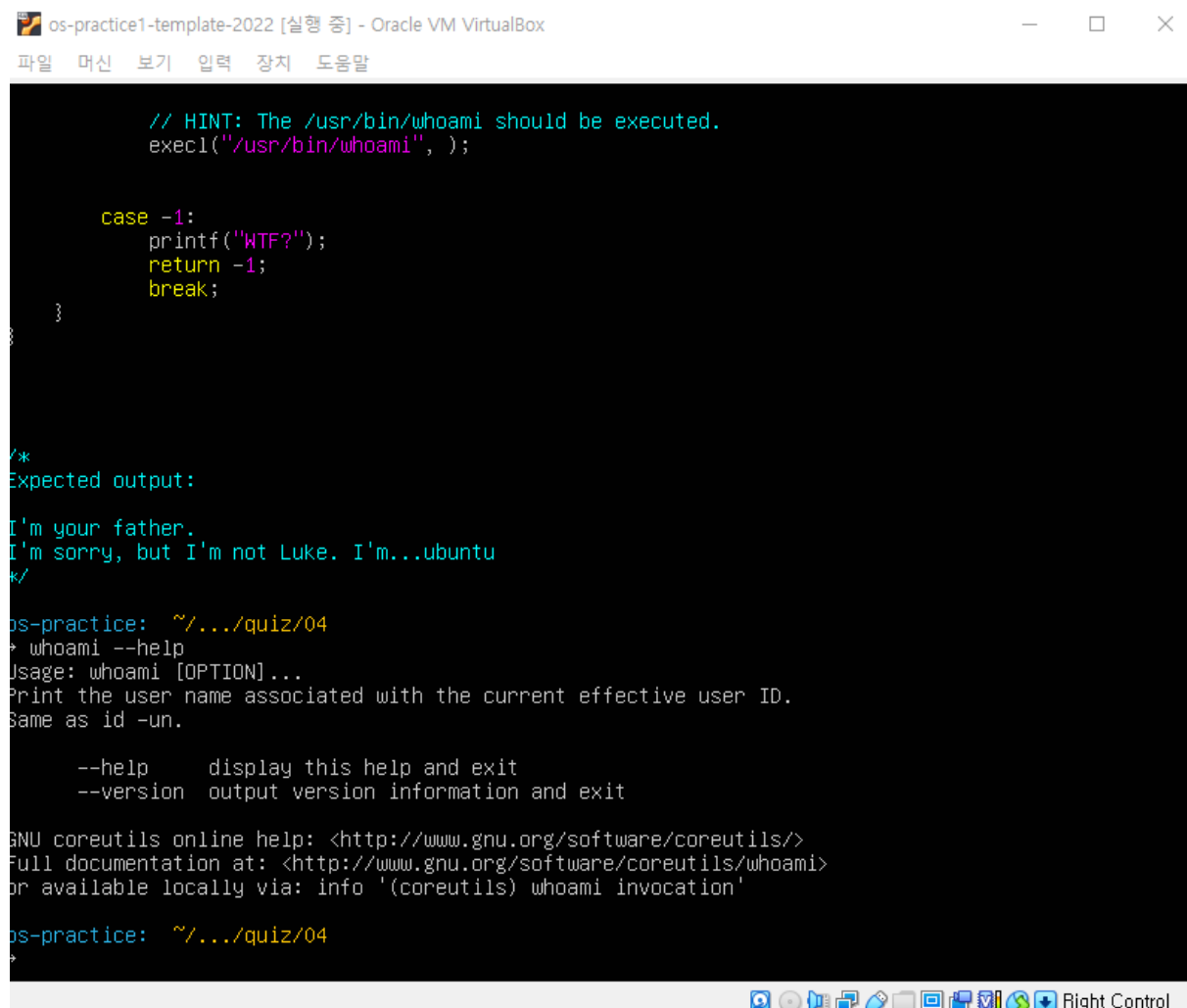
os-practice: ~/.../quiz/03
+ gcc -o main main.c
main.c: In function 'main':
main.c:17:5: warning: missing sentinel in function call [-Wformat=]
      execl("/bin/ls", "ls", "-l");
      ~~~~~

os-practice: ~/.../quiz/03
+ ls
main main.c

os-practice: ~/.../quiz/03
+ ./main
Before executing `ls -l`.
total 16
-rwxrwxr-x 1 guest guest 8344 Apr 12 02:17 main
-rw-rw-r-- 1 guest guest  602 Apr 12 02:16 main.c

os-practice: ~/.../quiz/03
+ -
```


프로세스 #04



```
// HINT: The /usr/bin/whoami should be executed.
execl("/usr/bin/whoami", );

case -1:
    printf("WTF?");
    return -1;
    break;
}

/*
Expected output:
I'm your father.
I'm sorry, but I'm not Luke. I'm...ubuntu
*/

os-practice: ~/.../quiz/04
$ whoami --help
Usage: whoami [OPTION]...
Print the user name associated with the current effective user ID.
Same as id -un.

    --help      display this help and exit
    --version   output version information and exit

GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Full documentation at: <http://www.gnu.org/software/coreutils/whoami>
or available locally via: info '(coreutils) whoami invocation'

os-practice: ~/.../quiz/04
$
```

/usr/bin/ 위치에서 whoami 명령어를 실행

이 때 whoami 명령어에 대해 알아보려고

Whoami -help를 통해서 알아보았다.

```
switch (pid)
{
    default:
        // HINT: The parent process should fall into this scope.
        printf("I'm your father.\n");
        sleep(3);
        break;

    case 0:
        sleep(1);
        // HINT: The child process should fall into this scope.
        printf("I'm sorry, but I'm not Luke. I'm...");
        fflush(stdout);

        sleep(1); // for dramatic effect

        // HINT: The /usr/bin/whoami should be executed.
        exec("/usr/bin/whoami", "whoami", NULL);

    case -1:
        printf("WTF?");
        return -1;
        break;
}

/*
Expected output:
I'm your father.
I'm sorry, but I'm not Luke. I'm...ubuntu
*/
-- INSERT --
```

33,52

Bot

```
// HINT: The child process should fall into this scope.
printf("I'm sorry, but I'm not Luke. I'm...");
fflush(stdout);

sleep(1); // for dramatic effect

// HINT: The /usr/bin/whoami should be executed.
execl("/usr/bin/whoami", "whoami", NULL);

case -1:
    printf("WTF?");
    return -1;
    break;
}

}

/*
Expected output:

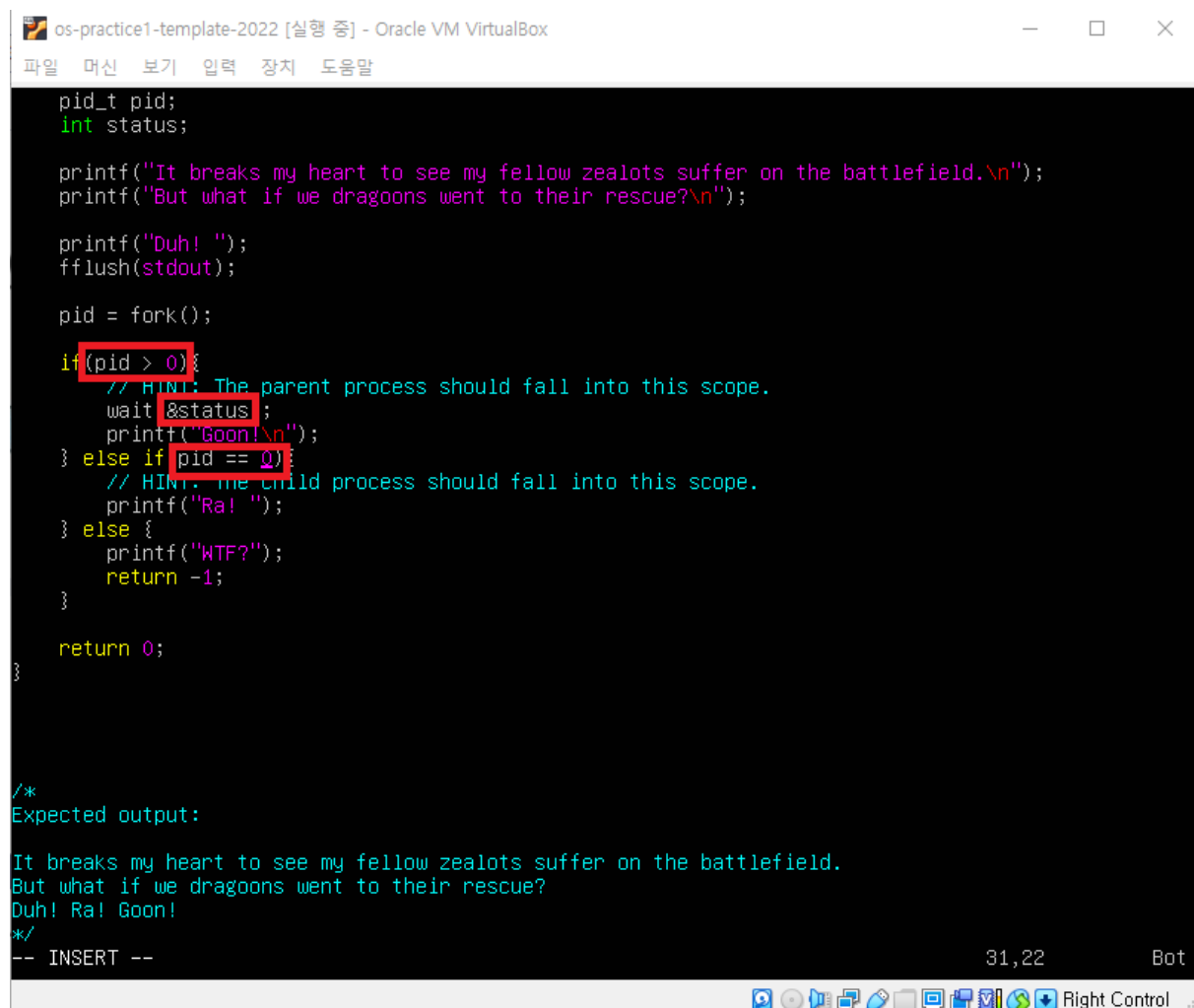
I'm your father.
I'm sorry, but I'm not Luke. I'm...ubuntu
*/

os-practice: ~/.../quiz/04
→ gcc -o main main.c

os-practice: ~/.../quiz/04
→ ./main
I'm your father.
I'm sorry, but I'm not Luke. I'm...guest

os-practice: ~/.../quiz/04
→ -
```

프로세스 #05



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

pid_t pid;
int status;

printf("It breaks my heart to see my fellow zealots suffer on the battlefield.\n");
printf("But what if we dragoons went to their rescue?\n");

printf("Duh! ");
fflush(stdout);

pid = fork();

if(pid > 0){
    // HINT: The parent process should fall into this scope.
    wait(&status);
    printf("Goon!\n");
} else if pid == 0{
    // HINT: The child process should fall into this scope.
    printf("Ra! ");
} else {
    printf("WTF?");
    return -1;
}

return 0;
}

/*
Expected output:
It breaks my heart to see my fellow zealots suffer on the battlefield.
But what if we dragoons went to their rescue?
Duh! Ra! Goon!
*/
-- INSERT --
```

31,22 Bot

Right Control

```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

if(pid > 0){
    // HINT: The parent process should fall into this scope.
    wait(&status);
    printf("Goon!\n");
} else if(pid == 0){
    // HINT: The child process should fall into this scope.
    printf("Ra! ");
} else {
    printf("WTF?");
    return -1;
}

return 0;
}

/*
Expected output:

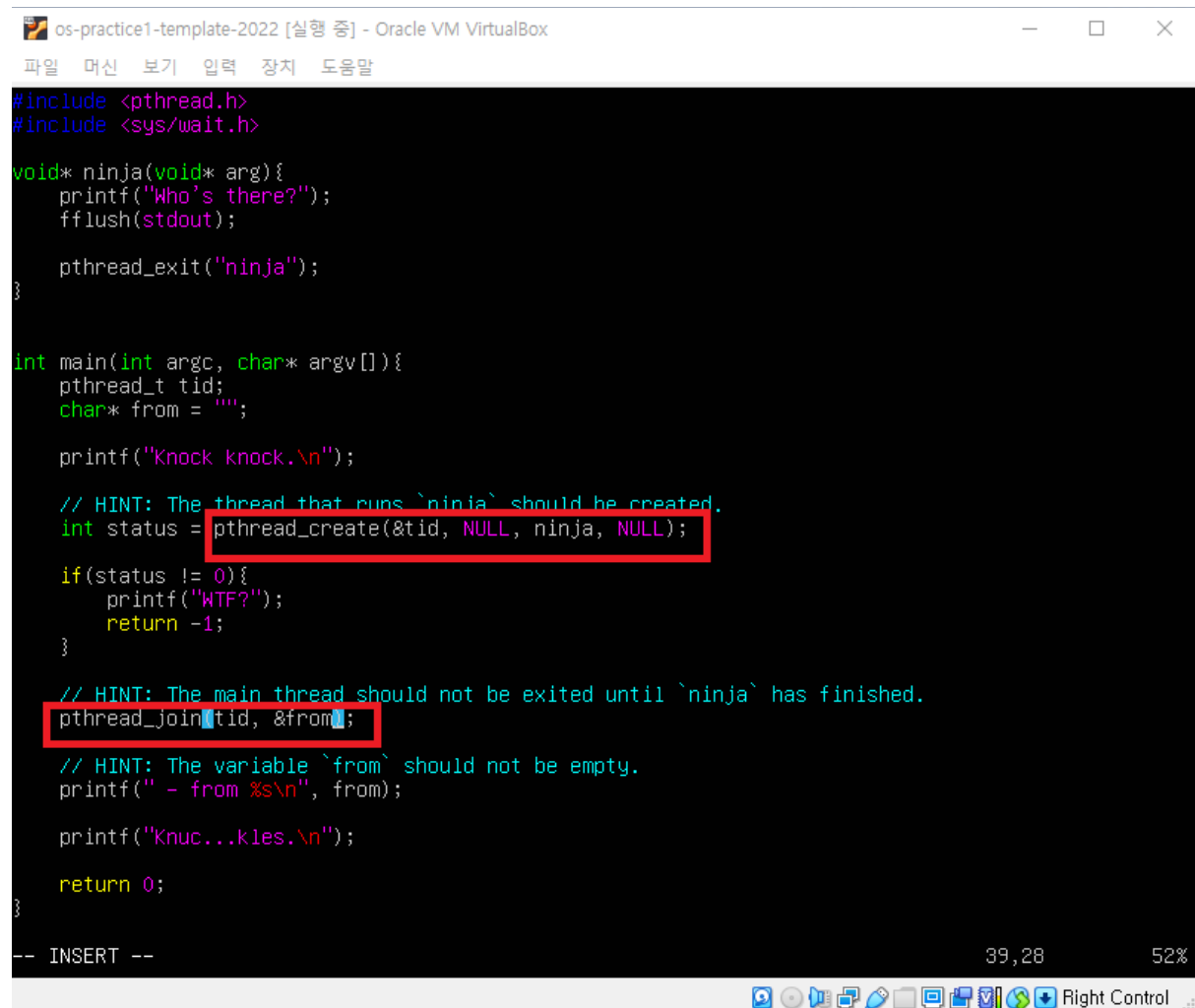
It breaks my heart to see my fellow zealots suffer on the battlefield.
But what if we dragoons went to their rescue?
Duh! Ra! Goon!
*/

os-practice: ~/.../quiz/05
+ gcc -o main main.c

os-practice: ~/.../quiz/05
+ ./main
It breaks my heart to see my fellow zealots suffer on the battlefield.
But what if we dragoons went to their rescue?
Duh! Ra! Goon!

os-practice: ~/.../quiz/05
+ -
```

스레드 #01



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

#include <pthread.h>
#include <sys/wait.h>

void* ninja(void* arg){
    printf("Who's there?");
    fflush(stdout);

    pthread_exit("ninja");
}

int main(int argc, char* argv[]){
    pthread_t tid;
    char* from = "";

    printf("Knock knock.\n");

    // HINT: The thread that runs `ninja` should be created.
    int status = pthread_create(&tid, NULL, ninja, NULL);

    if(status != 0){
        printf("WTF?");
        return -1;
    }

    // HINT: The main thread should not be exited until `ninja` has finished.
    pthread_join(tid, &from);

    // HINT: The variable `from` should not be empty.
    printf("- from %s\n", from);

    printf("Knuc...kles.\n");

    return 0;
}

-- INSERT --                                     39,28      52%
```

```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

    if(status != 0){
        printf("WTF?");
        return -1;
    }

    // HINT: The main thread should not be exited until `ninja` has finished.
    pthread_join(tid, &from);

    // HINT: The variable `from` should not be empty.
    printf(" - from %s\n", from);

    printf("Knuc...kles.\n");

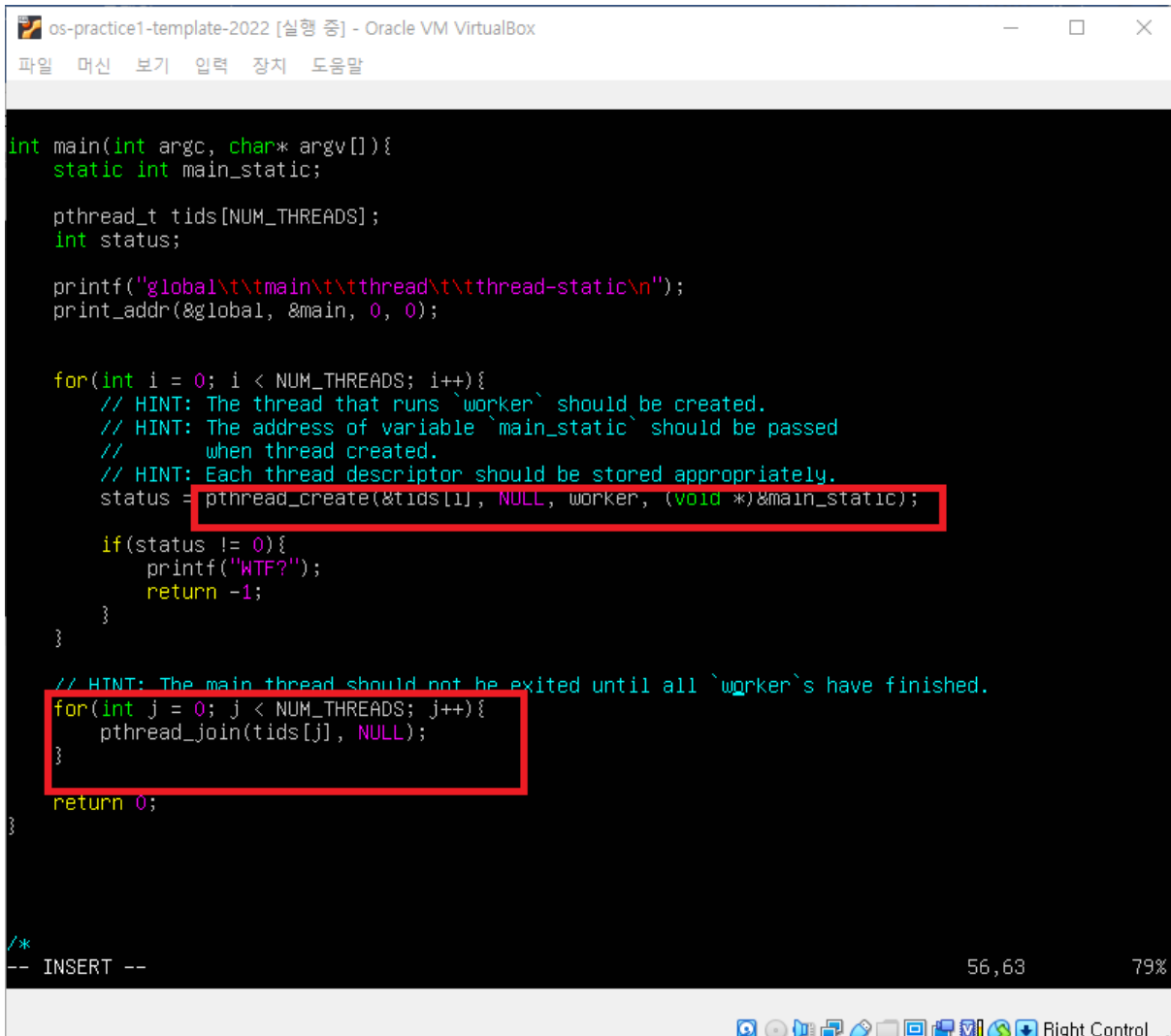
    return 0;
}

os-practice: ~/.../quiz/01
→ gcc -o main main.c -pthread
main.c: In function 'main':
main.c:39:23: warning: passing argument 2 of 'pthread_join' from incompatible pointer type [-Wincompatible-pointer-types]
    pthread_join(tid, &from);
                      ^
In file included from main.c:13:0:
/usr/include/pthread.h:251:12: note: expected 'void **' but argument is of type 'char **'
extern int pthread_join(pthread_t __th, void **__thread_return);
           ~~~~~

os-practice: ~/.../quiz/01
→ ./main
Knock knock.
Who's there? - from ninja
Knuc...kles.

os-practice: ~/.../quiz/01
→ -
```

스레드 #02



```
int main(int argc, char* argv[]){
    static int main_static;

    pthread_t tids[NUM_THREADS];
    int status;

    printf("global\t\tmain\t\tthread\t\tthread-static\n");
    print_addr(&global, &main, 0, 0);

    for(int i = 0; i < NUM_THREADS; i++){
        // HINT: The thread that runs `worker` should be created.
        // HINT: The address of variable `main_static` should be passed
        //       when thread created.
        // HINT: Each thread descriptor should be stored appropriately.
        status = pthread_create(&tids[i], NULL, worker, (void *)&main_static);

        if(status != 0){
            printf("WTF?");
            return -1;
        }
    }

    // HINT: The main thread should not be exited until all `worker`s have finished.
    for(int j = 0; j < NUM_THREADS; j++){
        pthread_join(tids[j], NULL);
    }

    return 0;
}

/*
-- INSERT --
```

56,63 79%

Right Control


```

// HINT: The address of variable `main_static` should be passed
//       when thread created.
// HINT: Each thread descriptor should be stored appropriately.
status = pthread_create(&tids[i], NULL, worker, (void *)&main_static);

if(status != 0){
    printf("WTF?");
    return -1;
}

// HINT: The main thread should not be exited until all `worker`s have finished.
for(int j = 0; j < NUM_THREADS; j++){
    pthread_join(tids[j], NULL);
}

return 0;
}

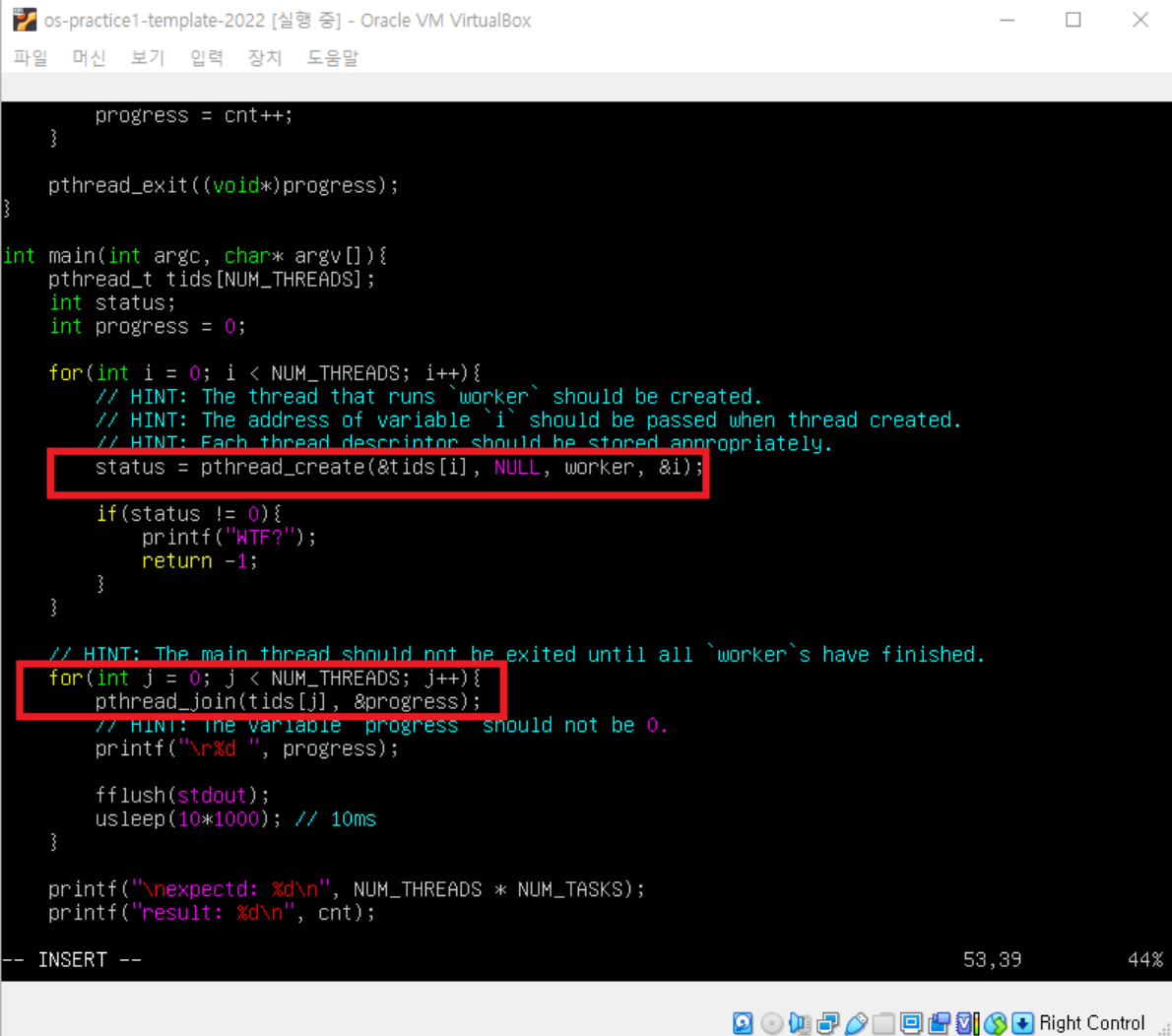
/*
os-practice: ~/.../quiz/02
→ gcc -o main main.c -pthread

os-practice: ~/.../quiz/02
→ ./main
global          main          thread          thread-static
0x56256dbaa014   0x56256d9a989f   (nil)   (nil)
0x56256dbaa014   0x56256dbaa01c   0x7fa4275e7ee4   0x56256dbaa018
0x56256dbaa014   0x56256dbaa01c   0x7fa426de6ee4   0x56256dbaa018
0x56256dbaa014   0x56256dbaa01c   0x7fa4265e5ee4   0x56256dbaa018

os-practice: ~/.../quiz/02
→ -

```

스레드 #03



```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

    progress = cnt++;
}

pthread_exit((void*)progress);
}

int main(int argc, char* argv[]){
    pthread_t tids[NUM_THREADS];
    int status;
    int progress = 0;

    for(int i = 0; i < NUM_THREADS; i++){
        // HINT: The thread that runs `worker` should be created.
        // HINT: The address of variable `i` should be passed when thread created.
        // HINT: Each thread descriptor should be stored appropriately.
        status = pthread_create(&tids[i], NULL, worker, &i);

        if(status != 0){
            printf("WTF?");
            return -1;
        }
    }

    // HINT: The main thread should not be exited until all `worker`s have finished.
    for(int j = 0; j < NUM_THREADS; j++){
        pthread_join(tids[j], &progress);
        // HINT: The variable `progress` should not be 0.
        printf("\r%d ", progress);

        fflush(stdout);
        usleep(10*1000); // 10ms
    }

    printf("\nexpected: %d\n", NUM_THREADS * NUM_TASKS);
    printf("result: %d\n", cnt);

-- INSERT --
53,39 44%
```

```
os-practice1-template-2022 [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말

main.c:31:18: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
    pthread_exit((void*)progress);
                  ^
main.c: In function 'main':
main.c:53:31: warning: passing argument 2 of 'pthread_join' from incompatible pointer type [-Wincompatible-pointer-types]
    pthread_join(tids[j], &progress);
                              ^
In file included from main.c:13:0:
/usr/include/pthread.h:251:12: note: expected 'void **' but argument is of type 'int *'
    extern int pthread_join(pthread_t __th, void **__thread_return);
               ~~~~~
os-practice: ~/.../quiz/03
→ ls
main  main.c
os-practice: ~/.../quiz/03
→ ./main
1388478
expectd: 10000000
result: 1945594
os-practice: ~/.../quiz/03
→ ./main
9399999
expectd: 10000000
result: 10000000
os-practice: ~/.../quiz/03
→ ./main
9099999
expectd: 10000000
result: 10000000
os-practice: ~/.../quiz/03
→ -
```